PERFORMANCE GAINS FROM LEAVING IDLE PROCESSORS IN MULTIPROCESSOR SYSTEMS

E. Smirni*, E. Rosti†, G. Serazzi‡, L.W. Dowdy*, K.C. Sevcik§
esmirni@vuse.vanderbilt.edu
rose@dsi.unimi.it
serazzi@ipmel2.elet.polimi.it
dowdy@vuse.vanderbilt.edu
kcs@csri.toronto.edu

Abstract

When a parallel job arrives at a multiprocessor and there are idle processors, it is a common practice to assign all available processors to the job. In this paper the performance advantages of not allocating all available processors in multiprocessor systems are investigated. The class of adaptive space-sharing policies is considered. Analytical models of simple adaptive policies are examined. Complex policies are presented and investigated by means of experiments on a 512 node Intel Paragon. For all experiments, real parallel applications are used as the system workload. Results are reported for single and multiclass cases. Sensitivity analysis with respect to the workload interarrival time distribution is presented.

Performance gains derived from saving computational power (i.e., leaving idle processors while there are parallel jobs in the waiting queue) are identified as a function of the system size and workload characteristics. The more variable the workload characteristics are, the higher the performance improvements. The results show that leaving processors idle is beneficial in multiclass situations or when the job arrival rate is quite variable.

1 MOTIVATION

Multiprogramming is a common way of improving performance of multiprocessor systems. Since few applications can take full advantage of all system processors and since it is not uncommon that several parallel jobs are waiting to use the multiprocessor, performance can improve by allocating a subset of the processors to each of several jobs. Software related issues such as maximum application parallelism, diminishing return from assigning additional processors, and variable speedup characteristics of various jobs, motivate different processor allocation strategies for multiprogramming parallel systems.

Several processor allocation strategies have been presented since 1982 [11, 7, 15, 18, 12, 6, 2, 19, 3, 8, 20, 9, 14, 13, 16, 1]. The proposed strategies differ from each other in the way they allocate resources to the executing programs. Each strategy is designed to work well under certain conditions. Depending upon the system architecture, the memory organization (i.e., shared or distributed), the topology of the interconnection network, and the parallel application characteristics, preemptive or non-preemptive policies are more appropriate. Under preemptive policies, parallel programs can be suspended while in execution to allow for processor redistribution according to the system load. Time-sharing policies and dynamic space-sharing policies are considered preemptive. Non-preemptive [17], adaptive [13, 14], or run-to-completion policies [19, 1] do not allow processor reallocation while a job is in execution. Each job is assigned to a limited set of processors (i.e., a partition). The number of processors per partition is usually a function of the system load (i.e., the arrival rate of parallel jobs) and the speedup characteristics of the arriving jobs. The higher the workload arrival rate, the smaller the partition size should be to guarantee low average job response time and high system throughput [13, 14, 1].

The low implementation cost of non-preemptive policies makes them particularly appropriate for distributed memory message passing architectures [17]. However, non-preemptive policies are not as flexible as dynamic ones, which can adapt the number of allocated processors to each job according to system load fluctuations. Various solutions have been proposed to improve upon non-preemptive policy flexibility. A possible approach is to set aside some processors as a reserve for unpredictable changes in the workload intensity.

*Department of Computer Science - Vanderbilt University - Tennessee, USA.
†Dipartimento di Scienze dell'Informazione - Università di Milano, Italy.
‡Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy.
§Computer Systems Research Institute, University of Toronto, Canada.
This work was partially supported by Italian MURST 40% and 60% projects and by sub-contract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

In this paper, the relative benefits of restricting the number of processors assigned to a job under specific conditions are examined. Policies that keep some of the available processors idle for later allocations are investigated. Such an approach is somewhat counter-intuitive, since idle processors do not contribute to maximum system utilization. Several cases where preserving idle processors is beneficial to performance, are identified. In the single processor case, there is no advantage of keeping the processor idle. When multiple processors are available, assigning an additional processor to a job may not improve performance significantly and it may be better to allocate the processor to a potential future arrival. Thus, it may be better to save some processors when future is uncertain. As uncertainty increases, it is more important to keep some processors unallocated. As expected, the relative performance gains depend upon the workload characteristics.

Analytical models of simple two-partition policies are presented and solved. The impact on performance is investigated. Factors such as offered system load, workload arrival process, workload speedup and composition (single class versus multiclass) are considered. The analysis of more complex policies is performed experimentally on a 512 node Intel Paragon multiprocessor. An LU decomposition kernel is used as the test workload for the experiments.

This paper is organized as follows. Section 2 presents the results of the analysis of simple policies. Section 3 reports the results of the experimental study of complex policies. Section 4 summarizes the findings and concludes the paper.

2 SIMPLE POLICIES

In this section, Markovian models of simple two-partition policies and performance analysis based on their solution are presented. Simple policy models can be solved analytically and provide useful insights for more complex systems. Initially, interarrival and execution times are assumed to be exponentially distributed.

2.1 Policy Description

The policies considered in this section assign either all processors or half of the processors to a job, based on information available at scheduling time. As a base case, a policy that never keeps processors idle if there are jobs waiting for execution is considered. Processor Saving (PS) policies which do not always allocate all available processors are derived from the base case. A systematic way of constructing PS policies is suggested.

Figure 1 depicts the Markov diagrams of the base case policy (top diagram) and two different PS policies (two lower diagrams). A state is described by the notation (q, r_p) where q is the number of jobs awaiting processor assignment, r is the number of executing jobs, and p indicates the partition assigned to the executing jobs. If the job has been assigned half the system, then p is h. If the job is assigned the whole system, then p is w. State $(3, 1_w)$ indicates that there are 3 jobs waiting in queue, and 1 job is executing on the entire system. In state 0 the system is

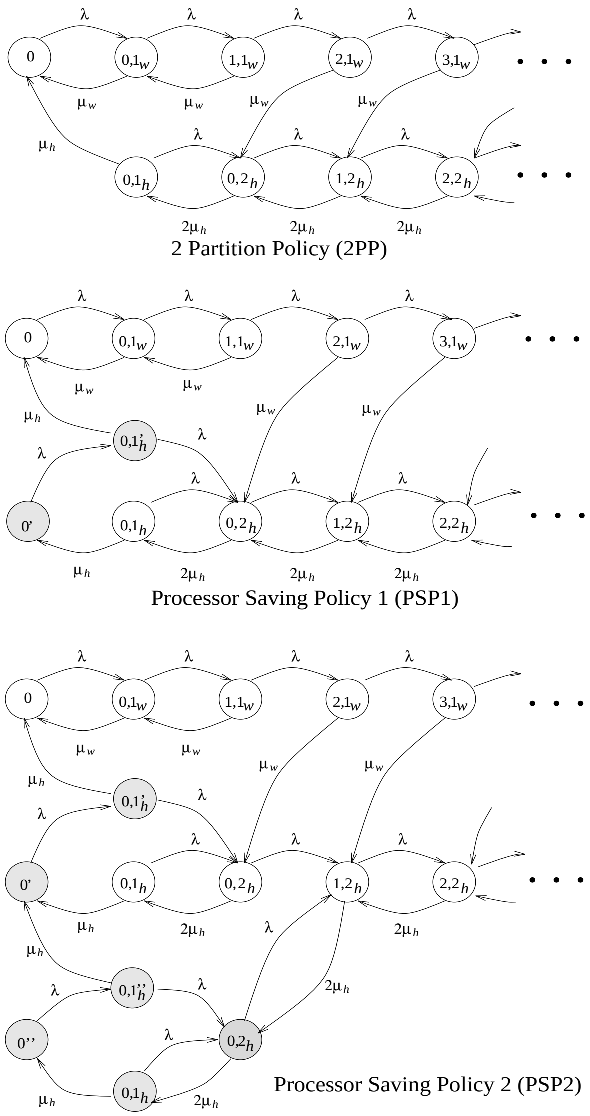

Figure 1: Markov diagrams of three allocation policies: 2PP, PSP1, and PSP2, which exhibit increasing amounts of processor saving.

completely idle. Shaded states indicate that a "processor saving" decision has been, or will be, made. λ and μ_i represent the job arrival rate and the job average execution rate on partition i, respectively. The base case 2-partition policy keeps all processors busy in presence of waiting jobs. The two PS policies are examples of possible strategies for saving processing power where in cases processors are kept idle, even though there are waiting jobs. In all cases, the policies are non-preemptive.

The base case 2-Partition Policy (2PP) (top diagram in

Figure 1) assigns the whole system to a single job if there is a single job requesting service in the system. Upon completion of a job that has been allocated the entire system, and if there are two or more jobs in the queue, the first two are scheduled, each being allocated half of the processors. With 2PP, at most two partitions are active, even when the waiting queue grows longer than 2.

The first Processor Saving policy PSP1 (middle diagram in Figure 1) is obtained by changing the scheduling decisions in the base case policy. When a job arrives at an empty system and the previously completed job had been allocated only half the processors (shaded $(0^{'})$ state), the next arriving job is only allocated half the processors (shaded $(0, 1_h^{'})$ state). Thus the system "remembers" that both its partitions were busy in the recent past. It decides to "save" processing power since the past states suggest that new arrivals are highly probable. If another job arrives before the newly scheduled one completes, it will find an immediately available partition. If no arrivals occur, a light load condition is inferred and the policy returns to the 2PP behavior (state (0)).

By increasing the amount of memory introduced in the policy, the return to the base case behavior can be further delayed. The bottom diagram of Figure 1 describes another processor saving policy (PSP2) derived by adding 4 states to the PSP1 case. Under PSP2, the system remembers that both partitions were recently busy with one waiting job (state $(1, 2_h)$). Because of this recent past, the next two arrivals will be assigned only half the processors instead of all processors. The justification is that it is likely that several jobs may arrive in the near future and saving processors for such a case is beneficial.

The procedure followed for constructing PSP2 can be applied systematically to generate new policies with even more memory levels than PSP2 and PSP1. The more "levels" of memory are introduced, the more slowly the policy returns to the original 2PP behavior. Introducing additional states in order to remember the recent past behavior is equivalent to considering higher order Markov chains.

2.2 Analytical Results

The models presented in the previous section are solved analytically using the global balance equations technique and performance figures are derived. The results are reported in this section. The impact of workloads with different speedups and various intensities (i.e., arrival rates to the system) is investigated. The performance metric adopted is the response time ratio [1], defined as the ratio of the average response time under a given policy to the average response time under the 2-Partition Policy. By varying the job arrival rate λ, the workload intensity is also varied. The system is studied under the range of utilization levels.

Figure 2 plots the response time ratio against system utilization under policy PSP1 (solid line) and PSP2 (dashed line) with a flat speedup workload ($\mu_w = \mu_h$), a

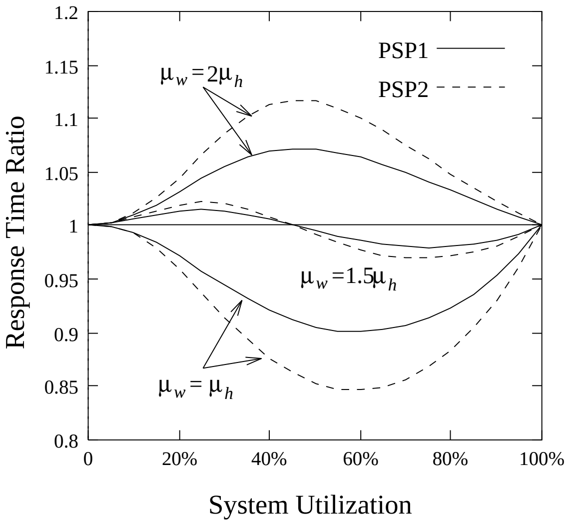

Figure 2: Response time ratio of the PS policies to the 2-Partition Policy (horizontal line) for three speedup workloads (linear, flat, and intermediate).

linear speedup workload ($\mu_w = 2\mu_h$), and with an intermediate speedup workload ($\mu_w = 1.5\mu_h$). Since the response time ratio (relative to the base case 2PP) is given, performance above the horizontal line at 1 indicates that it is better to always allocate all processors using 2PP. Similarly, performance below the horizontal line at 1 indicates that it is better to not allocate all processors and use a processor saving policy.

With a flat workload both PSP1 and PSP2 outperform 2PP. The maximum performance gain using PSP1 is 10% and the maximum performance gain using PSP2 is 15%. The PSP2 policy consistently outperforms PSP1 since it further delays the return to the 2PP type behavior. In the limit, with a flat workload, a fixed equipartitioning policy that assigns one processor to each job (i.e., an M/M/m server [5]), achieves the optimal performance. Such a policy exhibits a maximum performance gain over 2PP, since there is no advantage in assigning the entire system to an incoming job.

With a linear workload, there is no advantage in saving computational power. Performance loss results across the entire utilization range. This explains why PSP1 and PSP2 perform worse than the base 2PP policy in Figure 2.

As the speedup increases, spanning from flat to linear, the gain derived from PS strategies progressively reduces. Wave shaped response time ratios that intersect the 2PP line are obtained for workloads with intermediate speedups. The abscissa of the intersection depends on the workload speedup. The region to the left of the intersection point indicates the utilization interval where it is best to allocate all available processors. The region to the right of the intersection point indicates where it is best not to allocate all processors. As the system utilization increases, it is better to leave processors unallocated. As speedup increases, the intersection moves rightward, reaching 1 in the limit when the workload is nearly linear. Thus, as the speedup of the workload increases, it is better to allocate all processors as soon as possible. Nevertheless, since the difference between the total gains and losses for the two extreme speedups for a given PS policy is positive, if workload characteristics are not known, then using a PS policy improves performance more than it can hurt it. The potential maximum overall performance improvement due to a PS policy is greater than the potential maximum overall performance degradation.

As Figure 2 shows, for intermediate speedups, PSP1 performs better than PSP2 at low load, while their relative performance is reversed at higher load. At low load, keeping memory of long queues hurts performance since the system rarely returns to such states. A policy like 2PP that returns to larger partitions faster, following a temporary burst of job arrivals, performs better. On the contrary, at higher load, it is better not to allocate all processors after a temporary drop in job arrivals.

The results of the analysis of the simple policies considered in this section offer insight in the behavior of processor saving policies. The value of simple system modeling is the ability to isolate and focus on the peculiar aspects of the system under study abstracting the details which do not contribute to the observed phenomenon. When the analytical solution is computationally feasible, restricted models like the ones adopted here are useful for system analysis.

3 COMPLEX POLICIES

The policies presented in this section are more complex than those of Section 2. A wider variety of partition sizes is allowed. In the previous section, policies for which only two partitions are possible (whole and half) were examined analytically. In this section, more complex PS policies are considered for which analytical models are not feasible. Simulation and experimentation provide the alternative analysis techniques. An actual scheduler implementing the policies studied has been simulated on a real multiprocessor system. Actual applications were scheduled to arrive according to a Poisson arrival stream, which were then actually executed on the assigned partition.

The experimental platform is a 512 node Intel Paragon, a wormhole routed message passing multiprocessor system with distributed memory [4]. Results are presented for two system sizes, 32 and 64 processors. Experiments with 128, not reported here, confirm the findings of the smaller systems.

The implemented workload is an LU decomposition algorithm. It is executed on various matrix sizes, yielding different speedup curves. Four matrix sizes, namely 32×32, 64×64, 128×128, and 256×256, are considered. The corresponding speedup curves are reported in Figure 5.

A policy that never keeps processors idle in the presence of waiting jobs and a PS version derived from it have been implemented. The policies are described in the following section. The results of the base case experiment (i.e., with a Poisson arrival stream of single class jobs), are presented after the policy description. Sensitivity analysis with respect to the interarrival time distribution and experiments with multiclass workloads are also provided.

3.1 The Policies

The policies presented in this section allow for a wide variety of partition sizes. Partitions of various sizes provide greater flexibility to the workload fluctuations, thus improving the policy adaptability to unpredictable behavior. The goal is to mimic dynamic behavior using a non-preemptive policy thus avoiding the overhead of processor redistribution experienced by dynamic policies. With a wide variety of partition sizes, there are more opportunities for processor saving assignments, since the set of possible system states is larger. The pseudo-code algorithms of both the base case and the PS policies are given.

No Idle Processors Policy (NIP)
This policy is the base case of the experimental study and is analogous to the 2PP policy of Section 2. The NIP policy splits the entire processor set among the jobs waiting in queue for execution. Jobs are scheduled as long as there are free processors. When the number of free processors is smaller than the computed partition size, the job is scheduled on the remaining available processors. No information about the workload characteristics is needed.

Several policies have been presented in the literature that do not leave processors idle when there are pending requests [6, 3, 14, 1]. These policies divide the currently free processors among the waiting jobs, instead of the entire processor set. Such an approach is not adopted here in order to provide a uniform comparison with the PS policy considered. The partition size assigned to each job in this paper is computed as the total number of processors divided by the number of jobs in the queue. It has been demonstrated [13] that computing the partition size in this way results in better performance. The pseudo-code of the NIP algorithm is illustrated in Figure 3.

Processor Saving Adaptive Policy (PSA)
The PSA policy is based on a heuristic whose objective is to allocate an equal number of processors to all jobs in the system. Given the current state of the system, the new partition size is computed. If the number of free processors is smaller than the computed partition size, then no job is scheduled and the free processors are kept idle. In this case, a processor saving (i.e., non-work-conserving) decision is made, since some processors are kept idle while jobs are waiting for execution. The free processors will be assigned either: 1) if additional processors are released as jobs complete execution, or 2) if new jobs arrive, which force a smaller partition size. In the first case, a larger partition is granted at the expense of a longer waiting time.

```
job_in_queue := length(waiting_jobs_queue)
if (job_in_queue > 0) then {
    part_nb := min(system_size, job_in_queue)
    part_size := [system_size/part_nb + 0.5]
    while ((free_proc > 0) and
        (job_in_queue > 0)) do {
        if (part_size > free_proc) then
            part_size := free_proc
        partition := find(system, part_size)
        schedule(job, partition, part_size)
        }
}
```

Figure 3: No Idle Processors policy.

The potential gain or loss depends on the workload characteristics. If the waiting time for the partition size to become available is expected to be short and the waiting job can utilize the extra processors, then saving processors is beneficial. However, if the waiting time is long, or if the extra processors would not be efficiently utilized, or if additional jobs arrive which force a smaller target size, then it would be more beneficial to not keep available processors idle in the presence of waiting jobs. By adjusting the partition size, the PSA policy can vary the amount of processor saving that occurs. This is the key feature of the PSA policy. In the version analyzed here, memory of previous partition sizes is used. The motivation is that the partition size should change more slowly than the system state. A small partition size indicates many jobs in the system. If all jobs suddenly finish, it may not be wise to increase the partition size immediately in anticipation of another burst in job arrivals. Memory of the number of processors released by the last departing job (i.e., the size of the last released partition), is recorded in the variable last_released_part. A scheduling decision is made based upon this history variable and the current system state (i.e., the number of currently executing jobs (running_jobs), the number of free processors, and the number of partitions (part_nb)). No information about workload characteristics is needed. The pseudo-code of the algorithm is illustrated in Figure 4.

The PSA policy adapts to workload changes. Since no knowledge of the workload speedup characteristics is required to compute the partition size, the PSA policy is relatively robust. This policy has been introduced in [13], where the focus was on the policy robustness and the benefits of the processor saving aspects were not explored.

3.2 Experiments - Single Class Workload

In this section, experimental results on the Intel Paragon with various single class workloads for two system sizes, 32 and 64 processors, are presented. An LU decomposition algorithm is implemented and executed for four different matrix sizes. Each matrix size yields a different speedup curve (see Figure 5).

```
job_in_queue := length(waiting_jobs_queue)
if (job_in_queue > 0) then {
    if (job_in_queue = 1) and
    ((running_jobs = 0 and
        last_released_part ≠ system_size) or
    (running_jobs = 1 and part_size = system_size))
    then {
        if (part_nb > 2) then
            part_nb := part_nb - 1
        else
            part_nb := 2
    }
    else {
        if (job_in_queue ≥ part_nb) then
            part_nb := min(system_size, job_in_queue)
        elseif ((part_nb - running_jobs) > 1) then
            part_nb := part_nb - 1
    }
    part_size := [system_size/part_nb + 0.5]
    while (free_proc > 0 and job_in_queue > 0) do {
        partition := find(system, part_size)
        schedule(job, partition, part_size)
    }
}
```

Figure 4: Processor Saving Adaptive policy.

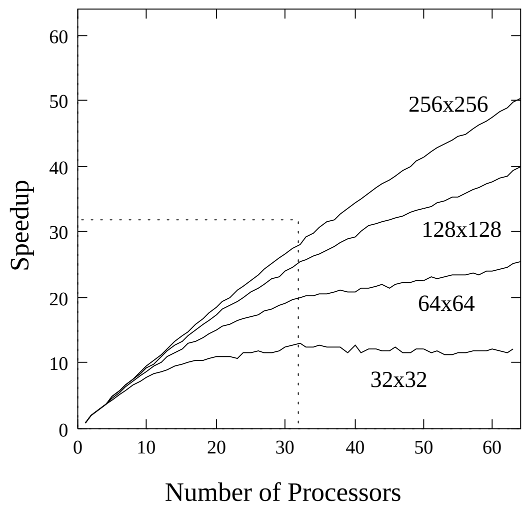

Figure 5: Speedup curves on the Intel Paragon for an LU decomposition application with various matrix sizes.

A Poisson stream of execution requests is submitted to the scheduler. The range of possible system utilizations is considered by changing the job arrival rate. Arriving requests join the waiting queue and the scheduler services them in FIFO order by allocating the computed number of processors (according to the policy) to the job at the head of the queue. Once a job is allocated a set of processors, the LU decomposition is executed. For every arrival rate considered, an arrival stream of four thousand jobs is submitted to the system and the average response time is measured.

The response time ratios of the PSA policy versus the NIP policy are plotted in Figure 6. The response time

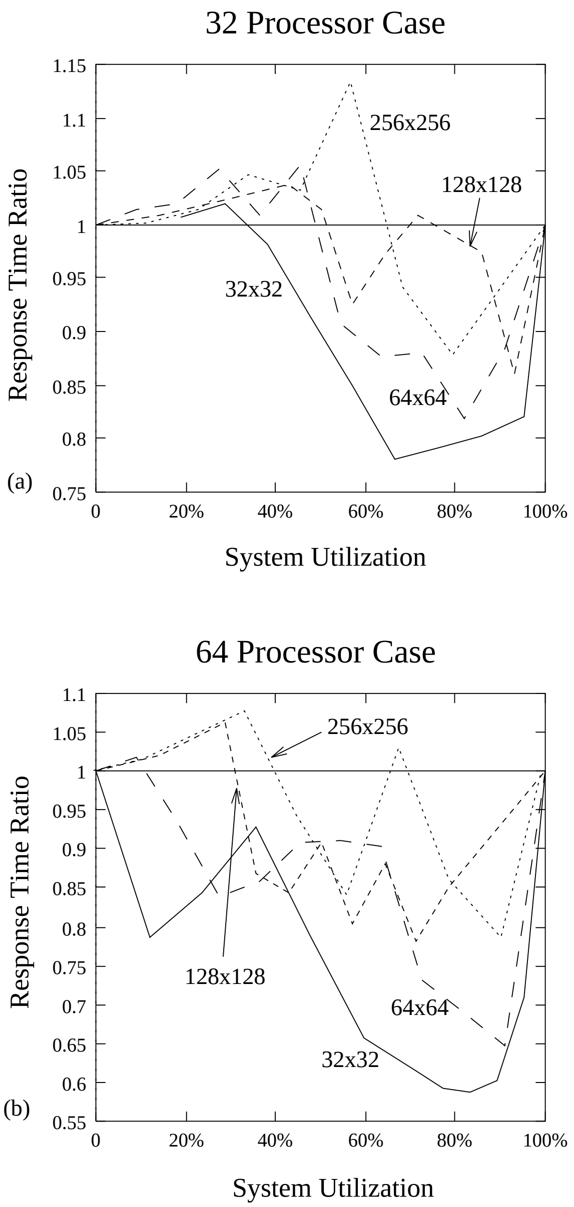

Figure 6: Response time ratios of the PSA policy to the NIP policy for the experiments on the Intel Paragon with 32 and 64 processors, for an LU decomposition application with various matrix sizes.

ratios follow the trend observed in the analytical results obtained in Section 2.2. For a system size of 32 (see Figure 6(a)), the maximum gain is obtained for the lowest speedup (i.e., the 32×32 case). The intersection point of the reference horizontal line and the PSA policy curve occurs at low system utilization and around 35%. As the speedup increases, the intersection point moves to the right. For the 256 case the intersection is around 65% of the system utilization. Because of measurement instabilities, the curves are not as smooth as desired[1]. Regardless of the spikes in the curves, there is a clearly observed trend.

In the 64 processor case, the response time ratios of the PSA policy are reported against the system utilization in Figure 6(b). Note the different y-axis scales between Figures 6(a) and 6(b). The performance gain of the PSA policy with the larger system is more significant. Although the span of the speedup curves is more spread than in the 32 processor case (see Figure 5), for the same speedup type, the PSA policy performs better as the system size increases. As an example, consider the speedup of the 32×32 matrix for 32 processors and the speedup of the 64×64 matrix for 64 processors. Both speedups exhibit almost identical behavior, showing the same efficiency (40%) for the maximum number of processors (32 and 64, respectively). However, the performance gains observed in the larger system are more pronounced. The intersection point is closer to the lower end of the utilization range. As the system size increases, the impact of wasted power due to not allocating processors as soon as possible is less severe. This is due to the fact that more jobs are in execution in larger systems. In a 2-partition system, where at most two jobs are executing, idle processors hurt performance more severely since the individual job response time has higher variance. The basic observation is that processor saving policies become more advantageous: 1) as the speedup behavior of the workload decreases, 2) as the system utilization increases, and 3) as the system size increases.

3.3 Sensitivity Analysis

In this section, the sensitivity analysis with respect to the interarrival time distribution for the single class workload is considered. Hyperexponential and hypoexponential interarrival times are examined. Results are reported for a single matrix size, 64×64, on 32 processors only. The results for larger system sizes and other matrix sizes are similar. Sensitivity analysis with respect to the application execution time distribution was not possible since a real workload with negligible variance in the execution time was used. Sensitivity analysis with real workloads with variance in the execution time will be the subject of future investigation.

[1]Even though each experiment was run multiple times, the average response times produced by each run were very close. It is possible that the jagged behavior of the curves will be alleviated by using different seeds in the exponential interarrival times for the incoming jobs, as well as different parallel workloads .

With hypoexponentially distributed interarrival times (i.e., with coefficients of variation less than 1), the variance in the interarrival times is smaller than with exponentially distributed interarrival times. The opposite is true with hyperexponential distributions, with coefficients of variation greater than 1. The hypothesis is that as the workload becomes more unpredictable (i.e., as the CV increases), the better it is to save processors to compensate for sudden workload changes in the future. Thus, as the CV increases, the PSA policy is expected to outperform a policy that allocates all processors as soon as possible. Likewise, if the arrival process is more regular and has smaller variance, sudden changes in the arrival process are not likely to occur, and all processors can be kept busy without hurting performance.

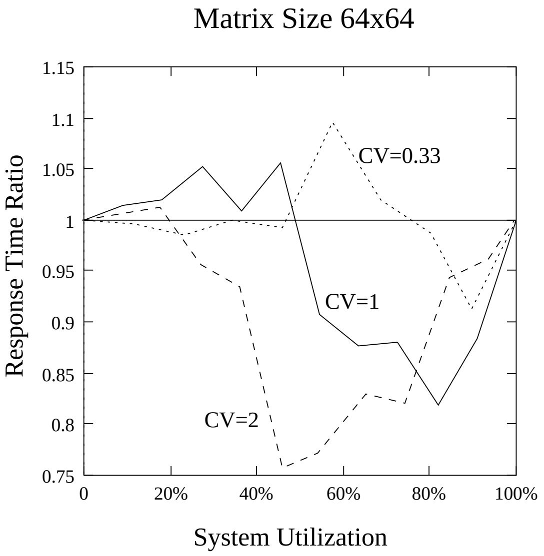

Figure 7: Response time ratios of the PSA policy to the NIP policy for the experiments on the Intel Paragon with 32 processors with hypoexponential and hyperexponential interarrival times as a function of the system utilization.

Figure 7 shows the response time ratios of the PSA policy against system utilization for coefficients of variation equal to 2 (dashed line), 1 (the base case, solid line), and 0.33 (dotted line). The experiments confirm that as the coefficient of variation increases, the intersection point moves to the left and that the curves drop. This indicates that keeping processors idle is more beneficial as the arrival process becomes more unpredictable.

3.4 Experiments - Multiclass Workload

In this section, multiclass workloads are considered. Multiclass workloads often are more representative of actual workloads since applications running on parallel systems vary widely with respect to execution requirements, scalability, and communication behavior. As in the previous section, the hypothesis is that as the workload becomes more unpredictable, meaning that more classes can be identified in the workload, the better it is to save some processors to respond to unpredictability.

The multiclass mixes considered are constructed from variable percentages of the single class components used in the single class analysis. Let the notation (W%, X%, Y%, Z%) represent a multiclass mix where W% of the arriving jobs perform LU decomposition on a matrix of size 32×32, X% of the arriving jobs solve a matrix of 64×64, Y% solve a matrix of size 128×128, and Z% solve a matrix of size 256×256. Since each matrix size leads to unique speedup characteristics (see Figure 5), the resulting workload exhibits a typical multiclass behavior.

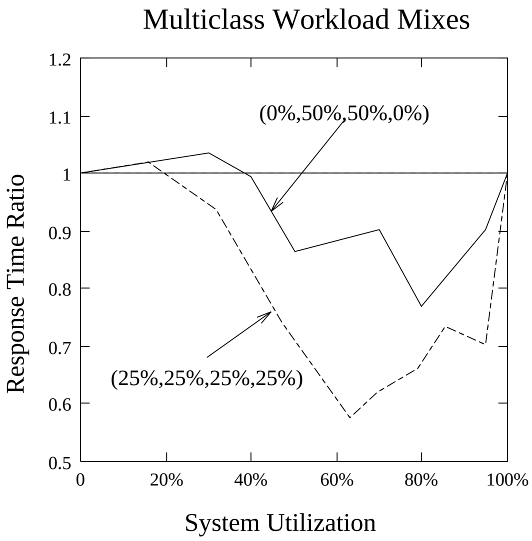

Figure 8: Response time ratios of the PSA policy to the NIP policy for the experiments on the Intel Paragon with 32 processors and two multiclass workload mixes.

In Figure 8, response time ratios for two workload mixes, namely (25%, 25%, 25%, 25%) and (0%, 50%, 50%, 0%) are reported. When the mix is balanced at (25%, 25%, 25%, 25%), the workload is the most heterogeneous. In this case, the intersection point is the leftmost and the benefit of saving processors is the highest. As the workload becomes more homogeneous and closer to single class, the intersection point moves to the right. These experiments support the hypothesis that if the workload components vary considerably, performance may improve by not allocating all processors as soon as possible.

4 CONCLUSIONS

In this paper, the advantage of not allocating processors to parallel applications, thus reserving some of the computational power for jobs yet to arrive, is investigated. Keeping some processors idle at the end of a scheduling round instead of allocating all processors whenever jobs are available to use them proves beneficial in certain cases:

- when the applications do not have good speedup characteristics,
- when the system utilization is medium to high,
- when the system size is large,
- when the job arrival rate is highly variable, or
- when the workload is multiclass.

The results of analytical models of 2-partition processor saving policies and of the experimental study of more complex policies support the above observations. Equivalently, the results indicate that if: 1) the workload speeds up linearly, 2) the multiprocessor system has low utilization, 3) the number of processors is small, 4) the job arrival is constant, and 5) the jobs are homogeneous, then allocating processors as quickly as possible is better.

Future work includes a more complete investigation of the impact of hyperexponential and hypoexponential distributions of the workload execution time and the workload arrival process. The effects of batch arrivals will be studied. A wider investigation of multiclass effects is planned. The usefulness of PS policies in the presence of processor failures will also be explored. The PSA policy will be compared with other processor saving policies presented in the literature. Experimental studies on various multiprocessor platforms are needed.

Acknowledgements
The authors would like to thank the Mathematical Sciences Research Section at Oak Ridge National Laboratory for providing access to the Intel Paragon xps/35 system.

References

[1] S.-H. Chiang, R.K. Mansharamani, M.K. Vernon, "Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies," *Proc. ACM SIGMETRICS*, 1994, pp. 33-44.

[2] K. Dussa, B.M. Carlson, L.W. Dowdy, K.-H. Park, "Dynamic partitioning in a transputer environment," *Proc. ACM SIGMETRICS*, 1990, pp. 203-213.

[3] D. Ghosal, G. Serazzi, S.K. Tripathi, "Processor working set and its use in scheduling multiprocessor systems," *IEEE Transactions on Software Engineering*, Vol 17(5), May 1991, pp. 443-453.

[4] Intel Corporation, **Paragon OSF/1 User's Guide**, 1993.

[5] L. Kleinrock, **Queueing Systems**, Vol 1, Wiley Interscience, 1975.

[6] S.T. Leutenegger, M.K. Vernon, "The performance of multiprogrammed multiprocessor scheduling policies," *Proc. ACM SIGMETRICS*, 1990, pp. 226-236.

[7] S. Majumdar, D.L. Eager, R.B. Bunt, "Scheduling in multiprogrammed parallel systems," *Proc. ACM SIGMETRICS*, 1988, pp. 104-113.

[8] S. Majumdar, D.L. Eager, R.B. Bunt, "Characterization of programs for scheduling in multiprogrammed parallel systems," *Performance Evaluation*, Vol 13(2), 1991, pp. 109-130.

[9] C. McCann, R. Vaswani, and J. Zahorjan, "A dynamic processor allocation policy for multiprogrammed shared memory multiprocessors," *ACM Transactions on Computer Systems*, Vol 11(2), February 1993, pp. 146-178.

[10] C. McCann, J. Zahorjan, "Processor allocation policies for message-passing parallel computers," *Proc. ACM SIGMETRICS*, 1994, pp. 19-32.

[11] J. Ousterhout, "Scheduling techniques for concurrent systems," *Proc. 3rd International Conference on Distributed Computing Systems*, 1982, pp. 22-30.

[12] K.-H. Park, L.W. Dowdy, "Dynamic partitioning of multiprocessor systems," *International Journal of Parallel Programming,* Vol 18(2), 1989, pp. 91-120.

[13] E. Rosti, E. Smirni, L.W. Dowdy, G. Serazzi, B.M. Carlson, "Robust partitioning policies for multiprocessor systems," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 141-165.

[14] S.K. Setia, M.S. Squillante, S.K. Tripathi, "Processor scheduling in multiprogrammed, distributed memory parallel computers," *Proc. ACM SIGMETRICS*, 1993, pp. 158-170.

[15] K.C. Sevcik, "Characterization of parallelism in applications and their use in scheduling," *Proc. ACM SIGMETRICS*, 1989, pp. 171-180.

[16] K.C. Sevcik, "Application scheduling and processor allocation in multiprogrammed multiprocessors," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 107-140.

[17] E. Smirni, E. Rosti, L.W. Dowdy, G. Serazzi, "Evaluation of multiprocessor allocation policies," Tech. Report, Computer Science Dept., Vanderbilt University, Nashville, TN, (submitted for publication).

[18] A. Tucker, A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," *Proc. of the 12th ACM Symposium on Operating Systems Principles*, 1989, pp. 159-166.

[19] J. Zahorjan, C. McCann, "Processor scheduling in shared memory multiprocessors," *Proc. ACM SIGMETRICS*, 1990, pp. 214-225.

[20] S. Zhou, T. Brecht, "Processor pool-based scheduling for large-scale NUMA multiprocessors," *Proc. ACM SIGMETRICS*, 1991, pp. 133-142.